VLSI synthesis of digital application specific neural networks
by Grant Philip Beagles

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering
Montana State University
© Copyright by Grant Philip Beagles (1992)

Abstract:
Most neural net research may be roughly classified into two general categories: software simulations or programmable neural hardware. The major drawback of any software simulation is that, no matter how well written, it is run on a sequential machine. This means that the software must simulate the parallelism of the network. Hardware neural networks are usually general purpose. Depending on the training, a significant percentage of the total hardware resources may be unused.

By defining a software model and then synthesizing a network from that model, all of the silicon area will be utilized. This should result in a significant reduction in chip size when comparing the application specific version to a general purpose neural network capable of being trained to perform the same task.

The purpose of this project is to synthesize an application specific neural network on a chip. This synthesis is accomplished by using parts of both categories described above. The synthesis involves several steps: 1) defining the network application; 2) building and training a software model of the network; and 3) using the model in conjunction with hardware synthesis tools to create a chip that performs the defined task in the expected manner.

The simulation phase of this project used Version 2.01 of NETS written by Paul T. Baffes of the Lyndon B. Johnson Space Center. NETS has a flexible network description format and the weight matrix may be stored in an ASCII file for easy usage in later design steps.

Hardware synthesis is done using the OCT CAD tools from U.C. Berkeley. The weight matrix (obtained from NETS) is translated into the OCT hardware description language (BDS) with several programs written explicitly for this project. OCT has several modules that allow a user to perform logic reduction, test the resulting logic circuit descriptions, and prepare the logic circuits for fabrication.

An application specific neural network, synthesized using the methodology described herein, showed a 75% reduction in total required silicon area when compared to a general purpose chip of the appropriate size. By using a software model to define an application specific neural network before hardware fabrication, significant savings in hardware resources will result.

VLSI SYNTHESIS OF DIGITAL APPLICATION SPECIFIC NEURAL NETWORKS

by

Grant Philip Beagles

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 1992

© COPYRIGHT

by

Grant Philip Beagles

1992

All Rights Reserved

N378
B357

APPROVAL

of a thesis submitted by

Grant Philip Beagles

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

April 6, 1992
Date

Chairperson, Graduate Committee

Approved for the Major Department

April 9, 1992
Date

Head, Major Department

Approved for the College of Graduate Studies

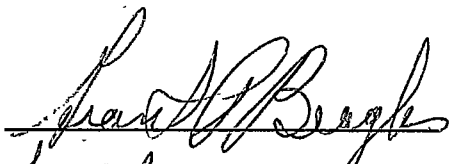
April 22, 1992
Date

Henry L Parsons
Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made.

Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted by the copyright holder.

Signature 

Date April 14, 1992

TABLE OF CONTENTS

TABLE OF CONTENTS - Continued

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES - Continued

ACKNOWLEDGEMENTS

This work was supported by an educational grant from the National Science Foundation MOSIS program and equipment donations from the Hewlett-Packard Co., Tektronix, Inc., and Advanced Hardware Architectures, Inc. I would especially like to thank Kel Winters and Paul Cohen of Advanced Hardware Architectures, and Dr. Gary Harkin, Jaye Mathisen, Diane Mathews and Bob Wall of Montana State University for their invaluable assistance.

ABSTRACT

Most neural net research may be roughly classified into two general categories: software simulations or programmable neural hardware. The major drawback of any software simulation is that, no matter how well written, it is run on a sequential machine. This means that the software must simulate the parallelism of the network. Hardware neural networks are usually general purpose. Depending on the training, a significant percentage of the total hardware resources may be unused.

By defining a software model and then synthesizing a network from that model, all of the silicon area will be utilized. This should result in a significant reduction in chip size when comparing the application specific version to a general purpose neural network capable of being trained to perform the same task.

The purpose of this project is to synthesize an application specific neural network on a chip. This synthesis is accomplished by using parts of both categories described above. The synthesis involves several steps: 1) defining the network application; 2) building and training a software model of the network; and 3) using the model in conjunction with hardware synthesis tools to create a chip that performs the defined task in the expected manner.

The simulation phase of this project used Version 2.01 of NETS written by Paul T. Baffes of the Lyndon B. Johnson Space Center. NETS has a flexible network description format and the weight matrix may be stored in an ASCII file for easy usage in later design steps.

Hardware synthesis is done using the OCT CAD tools from U.C. Berkeley. The weight matrix (obtained from NETS) is translated into the OCT hardware description language (BDS) with several programs written explicitly for this project. OCT has several modules that allow a user to perform logic reduction, test the resulting logic circuit descriptions, and prepare the logic circuits for fabrication.

An application specific neural network, synthesized using the methodology described herein, showed a 75% reduction in total required silicon area when compared to a general purpose chip of the appropriate size. By using a software model to define an application specific neural network before hardware fabrication, significant savings in hardware resources will result.

CHAPTER 1

INTRODUCTION

The Project

Most efforts in neural network research are easily classified in to two general categories. The first is software solutions and the second being hardware solutions. Software solutions involve modeling the functionality of a neural network on a sequential computer with simulation programs. The hardware solution entails general purpose neural network chips which are taught to perform a task. This project involves the union of these two categories into a methodology by which simple application specific neural hardware may be made.

The first step in the synthesis process is to define the problem. In the case of this project, a chip capable of recognizing the digits 0 through 9 was specified. The network would output the appropriate ASCII code for each numeral.

Form often follows functionality and this was definitely the case here. A number of factors determined the network configuration. The first was to limit, as much as possible, the number of nodes in the network. The input layer needed to have enough cells to allow reasonably clear definitions of the characters. The output layer is seven neurons since ASCII codewords are seven bits long. The hidden layer was constrained

on the low end by the need for a sufficient number of cells to successfully recognize the characters and on the high end by the need to limit the size of the chip that will eventually be defined.

Defining the problem also involved defining the training set to be used in teaching the model and developing evaluation criteria for the results. The training set used for this project utilized only "ideal" characters. No distortion or corruption of the numerals was considered to simplify the testing of the application specific approach. The simulator that was used simulates an analog neural network. The output values were not the digital "0" or "1" but values ranging from 0 to 1. The decision boundary used in the determination of the bit value was 0.5 with a "1" being 0.5 or greater.

The initial network was fully connected. Many of the connections contributed little or nothing to the solution of the problem. The next step in the process was to eliminate those connections. The approach used for this was to eliminate any connection having a weight that was less than some predetermined level. After each "pruning" of the network connectivity, the simulator was used to verify continued satisfactory functionality of the network. Pruning was continued until the network began to exhibit inconsistent results. The reduction level used was the largest one that still gave adequate separation between ones and zeros.

This is the point where the crossover from the software domain to hardware synthesis occurs. The weight matrix from the model was translated into a hardware description language (HDL) that can be used by the OCT CAD tools which are described in Chapter 4. A translator was written to automate this step.

A driving force behind this method is to reduce the size of the die as far as possible. Silicon area is occupied by either standard cells or routing channels. The neural logic is essentially a sum and compare. If the number of bits used in the sum is reduced, the number of cells experiences a similar fate. In turn, the need for fewer cells will require less routing. Reduction was done by summing the weights and then looking at the two's compliment sign bit. A positive sum causes neural activity (output a 1) while a negative result sets the neural output to zero.

The OCT tools take the HDL description of the network and eventually produce the mask descriptions for hardware fabrication. The process is actually broken down into a number of discrete steps. The first step takes the HDL and converts it into the boolean expressions for the logic. The second step is to optimize the logic and map it into a library of standard cells. Step three is to verify the logic using a simulation tool included in OCT. Following successful verification, the "place and router" is used and the chip is prepared for fabrication. The simulation/verification step is performed again before actually releasing the chip to a foundry.

Introduction to Neural Networks

Neural Nets vs. Sequential Computers

Most everyone is familiar, at least abstractly, with the way in which Von Neumann machines are designed. A set of instructions is stored in memory and executed in a sequential manner.

For many years, these machines have been considered capable of doing anything to near perfection. "Many believe that considerations of speed apart, it is unnecessary to design various new machines to do various computing processes. They can all be done with one digital computer, suitably programmed for each case, the universal machine [Anderson,89]." Recently, however, the limits of this architecture are becoming more recognizable, and a radically different form of processor is again being studied. This new design is the **neural network**.

A neural network is a massive parallel array of simple processing units which simulate (at least in theory) the neurons contained in the human brain. An electronic neural network attempts to mimic the ability of the brain to consider many solutions to a problem simultaneously. It has also been shown to have the ability to successfully process corrupted or incomplete data [Murray,88].

Until recently, existing high speed digital supercomputers have been programmed to simulate the workings of a neural network. With the advent of VLSI technology and the possibility of ULSI (ultra large scale integration) in the near future, it has become possible to realize fairly complex neural networks in silicon.

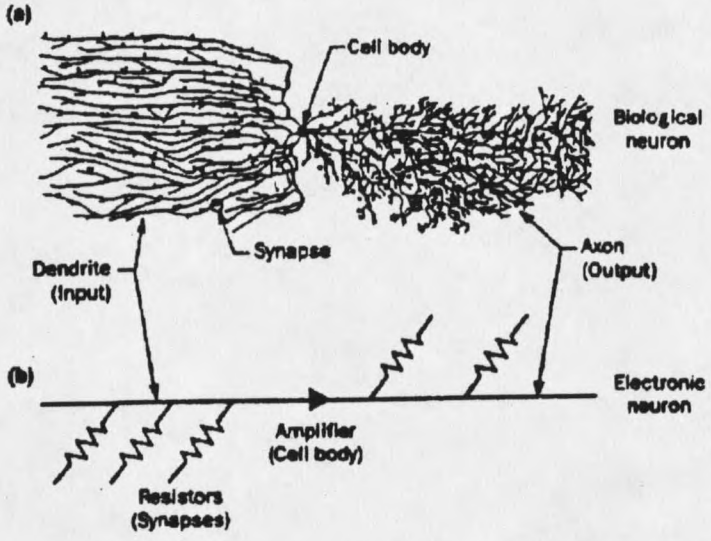
To date, electronic neural networks are still quite limited. Existing neural network hardware falls roughly into two categories: 1) one of a kind experimental projects; or 2) general purpose neural network chips that must be "taught".

Two examples that fall into the first category come from Bell Labs and Bellcore. A chip having 54 "neurons" manufactured by Bell Laboratories in Holmdel, New Jersey was "taught" to recognize key features in handwritten characters. This simple network was able to perform this task about 1,000 times faster than a suitably-programmed VAX 11/750 [Howard,88]. Bellcore has produced a neural network chip which they claim is 100,000 times faster than a neural network simulated on a general purpose computer [Cortese,88].

Neurons

The Biological Neuron A biological neuron and a simple electronic model of it taken from Richard E. Howard, et. al. are shown in Figure 1. The human brain contains about 10^{11} of these neurons, each with approximately 10^{4} input and output connections. Both the input and output structures are a branching tree of fibers. The input structures called ***dendrites*** receive incoming signals from other neurons through adjustable

connections known as ***synapses***. The output is sent through the ***axons*** to the synapses of other neurons.



Biological neuron and an electronic analog.
Figure 1

Intercellular communication is accomplished through streams of electric pulses of varying rates sent out along a cell's axons to the synapses of other cells. Some synapses are excitatory, that is, input along these channels tend to make the neuron increase its activity, while others are inhibitory and cause the cell to slow its output pulse rate [Howard,88].

The real heart of the scheme is that the synaptic connections are "adjustable," meaning that the signal received at each connection is weighted and the neural response

is based on a weighted sum of all inputs. This weighing is the end result of learning and is being constantly retuned.

The Electronic Neuron The electronic neuron, like its biological cousin, has a great number of weighted interconnections. In the electronic model, however, there are at best, thousands of connections, which is several orders of magnitude less than the biological network. Each neural response is based on a weighted sum of its inputs combined with a neuron threshold value to determine the neuron's level of activity.

Ideally, the neural network should be asynchronous, unlike the bulk of present day computers. There are valid arguments that point to an analog processor as being superior to a digital model.

In their paper, Murray and Smith present a brief contrast of the strengths and weaknesses of digital and analog networks which is shown below [Murray,88].

A. *Digital Neural Networks*

The strengths of a digital approach are:

- design techniques are advanced, automated, and well understood;
- noise immunity is high;
- computational speed can be very high; and
- learning networks (i.e., those with programmable weights can be implemented readily.

For neural networks there are several drawbacks:

- digital circuits of this complexity must be synchronous, while real neural nets are asynchronous;
- all states, activities, etc. in a digital network are quantized; and
- digital multipliers, essential to the neural weighting function, occupy large silicon area.

B. *Analog Neural Networks*

The benefits of analog networks are:

- asynchronous behavior is automatic;
- smooth neural activation is automatic; and
- circuit elements can be small.

The disadvantages include:

- noise immunity low;
- arbitrarily high precision is not possible; and
- worst of all, no reliable analog nonvolatile memory technology exists.

The weights which are set through a learning process must be stored somewhere. As stated above, digital networks can easily use memory to hold the weight values, and auxiliary support logic can be designed to "reteach" the network every time it is powered up. This also allows a single general purpose neural network to be used for more than one application. Analog technology falls somewhat short of the mark.

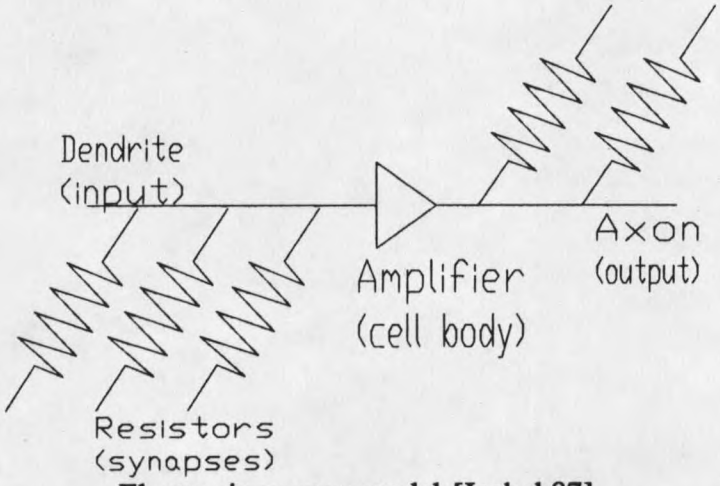
Figure 1 contains a representation of an electronic neuron which is shown in Figure 2. The neuron body is composed of an amplifier. Weighting of the synaptic connections is represented by resistors on both the input (dendrite) path and output (axon).

This model appears overly simplified, but it is really quite complete Several working neuron designs are nearly this simple.

The goal is not to identically imitate the biological neuron. Clearly, the models are grossly simplified compared to the genuine article, yet even these relatively simple networks have complex dynamics and show interesting collective computing properties. The model which is used to exploit these collective properties is referred to as a ***connectionist model*** [Graf,88]. In a connectionist model individual neurons do very little individual computations. It is the multiple parallel paths that allow the neural network to render astonishing results.

The abbreviated schematic in Figure 3 (page 12) is a portion of a circuit implemented by Bell Labs in Holmdel, New Jersey. The entire network consisted of 54

amplifiers with their inputs and outputs interconnected through a matrix of resistive coupling elements. All of the coupling elements are programmable; *i.e.*, a resistive connection can be turned on or off [Baird,89].



Electronic neuron model [Jackel,87].
Figure 2

"Teaching" the Net

All of the garden variety digital microprocessors in use today have one major trait in common; they need a built-in instruction set so that their users may program them. These instruction sets may be as succinct as a few tens of instructions or be composed of hundreds of complex instructions.

Neural networks have no built-in instruction sets. Programming a neural network is accomplished through a teaching process. Presently, there are three main methods of teaching neural networks, which are shown below.

1. Explicit programming of connectivity and weight assignments.
2. Learning algorithms that can be used to train a network starting from some (generally random) initial configuration.
3. Compiling approaches that take an abstract specification and translate it into a network description [Jones,88].

Explicit programming of connectivity and weight assignments requires that the problem be modeled and the initial weights be assigned so that the network exhibits the desired properties. Use of this method requires that some educated guessing takes place with regard to network configuration and connection weighting.

The major drawback to the learning algorithm method is the difficulty of designing a good training set. For example, the United States Postal Service has a set that is used to train optical readers that sort mail. This set, although limited to alpha-numeric characters, requires over ten thousand samples.

The teaching method most often used for this type of network training is back propagation. An input vector is applied to the input layer of a neural network and the network processes the vector. The network output is compared to the desired resultant vector and the error is propagated backwards through the network and the synaptic weights are adjusted. By using an iterative process, these learning algorithms can be used to adjust the neural weighting scheme until the error is within an acceptable range.

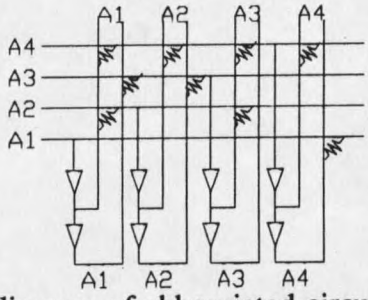
This method has the advantage of allowing greater flexibility in training the network. The network learns a task in the manner that is best suited to its design. Preconceived notions of how the network "should" function do not skew results. It should be noted that this is the technique employed in this project.

The third method involves programming the processor in somewhat the same way that a conventional machine is programmed. A neural network compiler would be necessary to convert a programmer's instructions into a language that the network could understand. The difficulties inherent in this method are the traditional notions of sequential operation.

Interconnection

In many designs, the resistive interconnection values are precalculated and put right into the silicon when the chip is made. Bell Labs has designed a network that has 256 individual neurons on a single chip, a portion of which is shown schematically in Figure 3. The connections are provided by amorphous-silicon resistors which are placed on the chip in the last fabrication step [Graf,86]. The end product of this type of chip synthesis is application specific; it will only perform a single task. Most chips of this type are one-of-a-kind experimental devices.

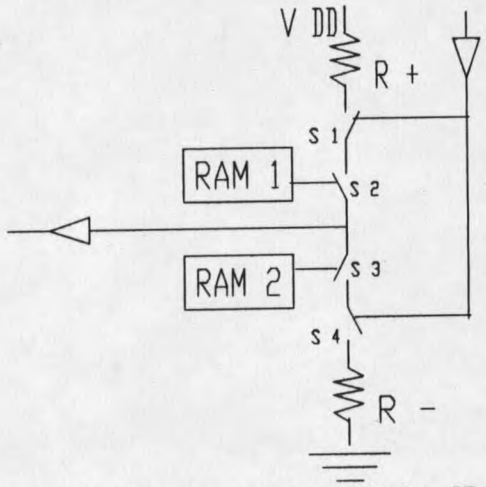
This method is by far the easiest to make and use, although it does not allow for a great deal of flexibility. However, since the weights are nonvolatile, reprogramming is not necessary every time the chip is powered off and on.



Schematic diagram of abbreviated circuit [Graf,88].

Figure 3

The resistors in Figure 2 were actually realized using the circuit in Figure 4. The output lines do not actually feed current into the input lines but instead control switches S1 and S4. The two memory cells determine the type of connection. One of three connections can be selected: 1) an excitatory (S2 enabled): 2) an inhibitory (S3 enabled); or 3) an open (both disabled) state [Graf,88].



RAM-controlled interconnection weights [Baird,89].

Figure 4

By employing the RAM cells into each neuron, it becomes a simple task to "teach" the network to perform a task, download the weighting information to nonvolatile storage, and reload it the next time the network is powered up. This method also allows the network to have numerous predetermined weighting patterns loaded for specific assignments. Although this general purpose approach may seem advantageous, the application specific approach may be more desirable.

A fascinating development in neuron theory comes from a model of the human immune response which in turn is modeled on Darwin's Theory of Evolution. The theory is that a neural network with hysteresis at the single cell level will more closely model a brain. The neuron is slightly more complicated than the model previously discussed. "Learning [in this type of network] occurs as a natural consequence of interactions between the network and its environment, with environmental stimuli moving the system around in an N-dimensional phase space until [an appropriate equilibrium is reached in response to the stimuli] [Hoffmann,86]."

In the connectionist model, the weighting is necessarily fixed, or at best, modifiable within strict limitations (usually -1, 0, +1). The hysteresis model does not have such restrictions. It should be able not only to learn, but to readily adapt to changing operational conditions. Experiments carried out at Los Alamos National Laboratories with this class of neuron have had promising results.

Technological Considerations

Several categories of electronic neurons are presently being employed in functional neural networks. The largest networks-on-a-chip existing today weigh in at around five hundred neurons. As feature sizes continue to shrink below the 1 μm range, 1000+ neuron networks on a single chip will become feasible [Sivilotti,86]. Such chips will be valuable tools in the direct modeling and evaluation of neural networks.

VLSI and ULSI chip design will allow for network processors containing a modest number of neurons. Silicon has an obvious drawback in realizing a network. Two dimensionality makes it impossible to build really large nets. Three dimensional biological material are more suited to this type of architecture, although the days of "growing" reliable processors is a long way off.

Neural network processing technology is in its infancy and no specific application for these processors has yet been clearly defined. It has even been shown that the optimal solution for many problems is in existing conventional equipment.

Neural networks have the ability to work with corrupted or incomplete data sets and to look at many solutions to a problem simultaneously. Due to this unique capability, network processors seem to be aptly suited to pattern recognition applications.

Conclusion

A biological neuron is a complex entity whose operation is only partially understood. Simple electronic models of a neuron can be built to mimic the basal operation of their biological counterpart. These simple processors are linked into connectionist networks which are capable of accomplishing tasks that are nearly impossible for most computers to do.

One thousand neuron networks are on the horizon. Although a quantum step backward from the 100 billion or so neurons of the human "network," the speed capability of a neural network chip far surpasses that of our brains. This differential will partially make up for the diminutive size and allow neural network processors to execute assignments with great dispatch. The day of the true "thinking machine" is far in the future, but the ***idiot savant*** is already here.

CHAPTER 2

NEURAL NETWORK MODELING

Neural Network Simulators

Many neural network simulators exist. It is even possible to simulate a network using any good spreadsheet program. Most neural net simulators that are available perform quite well. However, since only a simulation is involved, there are some drawbacks. Simulators are generally run on a sequential operation computer. A neural network by its very definition is a "highly parallel array of simple processors." The parallism must be modeled, which requires an iterative process. If the network being modeled has more than a few tens of nodes, or more than a few thousand connections, the number of iterations becomes astronomical.

NETS Neural Network Simulator

The simulator that is being used for this project is Version 2.01 of NETS written by Paul T. Baffes of the Software Technology Branch of the Lyndon B. Johnson Space Center. This simulator was chosen for several reasons. NETS has a flexible network description format which allowed easy modifications to a network. The source code was

available and well documented; this allows modifications to be made directly on the simulator. Finally, the weight matrix generated during the teaching phase may be stored in an ASCII file for easy usage in later design steps.

Defining the Network

The function that a neural network will be used for will place constraints on its configuration. As a first design effort, a simple network with three layers and 67 neurons was defined. The network consists of a 5 by 6 node input layer, one hidden layer that is also 5 by 6 nodes , and a 1 by 7 node output layer. This geometry was driven by three major factors: 1) the simulations were done on a PC which slows to a near standstill if the network is too large; 2) the network was trained to recognize the ten numerals and output the appropriate ASCII code (hence, the 1 by 7 node output layer); and 3) the network, mapped into a standard cell library, needed to be small enough to be fabricated on a reasonable size die.

The network geometry is fully connected, that is, every node in a layer is connected to every node in the layer below it. For the network in this example there are 1110 connections. The NETS network description for this network is contained in Appendix A. Figure 5 shows a stylized block diagram of the network.